


```
PPPPPPPP      AAAAAA      SSSSSSSS      RRRRRRRR      EEEEEEEEEE      AAAAAA      DDDDDDDD      UU      UU      TTTTTTTTTT
PPPPPPPP      AAAAAA      SSSSSSSS      RRRRRRRR      EEEEEEEEEE      AAAAAA      DDDDDDDD      UU      UU      TTTTTTTTTT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PPPPPPPP      AA      AA      SSSSSS      RRRRRRRR      EEEEEEEE      AA      AA      DD      DD      UU      UU      TT
PPPPPPPP      AA      AA      SSSSSS      RRRRRRRR      EEEEEEEE      AA      AA      DD      DD      UU      UU      TT
PP      AAAAAAAAAA      SS      RR      RR      EE      AAAAAAAAAA      DD      DD      UU      UU      TT
PP      AAAAAAAAAA      SS      RR      RR      EE      AAAAAAAAAA      DD      DD      UU      UU      TT
PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      AA      AA      SSSSSSSS      RR      RR      EEEEEEEEEE      AA      AA      DDDDDDDD      UUUUUUUUUU      TT
PP      AA      AA      SSSSSSSS      RR      RR      EEEEEEEEEE      AA      AA      DDDDDDDD      UUUUUUUUUU      TT
                                     ....
                                     ....
                                     ....
                                     ....

LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLLLL      IIIIII      SSSSSSSS
```

```
1 0001 0 MODULE PASS$READ_UTIL ( %TITLE, 'Utility routines used by READ'
2 0002 0 IDENT = '1-001' ! File: PASREADUT.B32 Edit: SBL1001
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1
29 0029 1
30 0030 1 ++
31 0031 1 FACILITY: Pascal Language Support
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1
35 0035 1 This module contains utility procedures used by
36 0036 1 the numeric READ procedures.
37 0037 1
38 0038 1 ENVIRONMENT: User mode - AST reentrant
39 0039 1
40 0040 1 AUTHOR: Steven B. Lionel, CREATION DATE: 1-April-1981
41 0041 1
42 0042 1 MODIFIED BY:
43 0043 1
44 0044 1 1-001 - Original. SBL 1-April-1981
45 0045 1 --
46 0046 1
```



```

48 0047 1 XSBTTL 'Declarations'
49 0048 1
50 0049 1 PROLOGUE DEFINITIONS:
51 0050 1
52 0051 1
53 0052 1 REQUIRE 'RTLIN:PASPROLOG'; ! Externals, linkages, PSECTs, structures
54 0116 1
55 0117 1
56 0118 1 TABLE OF CONTENTS:
57 0119 1
58 0120 1
59 0121 1 FORWARD ROUTINE
60 0122 1 PASS$GET_UNSIGNED: JSB READ_UTIL, ! Get an unsigned string
61 0123 1 PASS$GET_INTEGER: JSB READ_UTIL, ! Get an integer string
62 0124 1 PASS$GET_REAL: JSB READ_UTIL, ! Get a real string
63 0125 1 PASS$GET_ENUMERATED: JSB_READ_UTIL, ! Get an enumerated value string
64 0126 1
65 0127 1 FIND_NON_BLANK: JSB_FIND_NON_BLANK; ! Find next non-blank character
66 0128 1
67 0129 1
68 0130 1 MACROS:
69 0131 1
70 0132 1 NONE
71 0133 1
72 0134 1 EQUATED SYMBOLS:
73 0135 1
74 0136 1
75 0137 1 LITERAL
76 0138 1 !+
77 0139 1 ! Character class codes used below for CLASSTAB.
78 0140 1 !-
79 0141 1
80 0142 1 iv = 0, ! Invalid character
81 0143 1 BT = 1, ! Blank or Tab
82 0144 1 DG = 2, ! Digit
83 0145 1 DP = 3, ! Decimal Point
84 0146 1 SI = 4, ! Sign
85 0147 1 EL = 5, ! Exponent letter
86 0148 1 LT = 6, ! Other letter, dollar and underscore
87 0149 1
88 0150 1 !+
89 0151 1 ! Aliases for class codes for use in routines.
90 0152 1 !-
91 0153 1
92 0154 1 CLASS_IV = iv,
93 0155 1 CLASS_BT = BT,
94 0156 1 CLASS_DG = DG,
95 0157 1 CLASS_DP = DP,
96 0158 1 CLASS_SI = SI,
97 0159 1 CLASS_EL = EL,
98 0160 1 CLASS_LT = LT;
99 0161 1
100 0162 1
101 0163 1 FIELDS:
102 0164 1
103 0165 1 NONE
104 0166 1

```

```

: 105      0167 1 ! OWN STORAGE:
: 106      0168 1 !
: 107      0169 1 !
: 108      0170 1 ! OWN
: 109      0171 1 !
: 110      0172 1 !
: 111      0173 1 !
: 112      0174 1 !
: 113      0175 1 !
: 114      0176 1 !
: 115      0177 1 !
: 116      0178 1 !
: 117      0179 1 !
: 118      0180 1 !
: 119      0181 1 !
: 120      0182 1 !
: 121      0183 1 !
: 122      0184 1 !
: 123      0185 1 !
: 124      0186 1 !
: 125      0187 1 !
: 126      0188 1 !
: 127      0189 1 !

    !+ The following table is used for determining the class of a particular
    ! character. Each of the first 128 characters is assigned a class code
    ! as listed above in the LITERAL section.
    !-

    CLASSTAB: VECTOR [128, BYTE] PSECT (_PASSCODE) INITIAL (BYTE(
        iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,BT,iv,iv,iv,iv,iv,iv, 00-0F
        iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv, 10-1F
        BT,iv,iv,iv,LT,iv,iv,iv,iv,iv,iv,SI,iv,SI,DP,iv, 20-2F
        DG,DG,DG,DG,DG,DG,DG,DG,DG,DG,iv,iv,iv,iv,iv,iv, 30-3F
        iv,LT,LT,LT,EL,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT, 40-4F
        LT,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,iv,iv,iv,iv,LT, 50-5F
        iv,LT,LT,LT,EL,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT, 60-6F
        LT,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT,iv,iv,iv,iv)); 70-7F

```

```
129 0190 1 %SBTTL 'PAS$$GET_UNSIGNED - Find an unsigned number string'
130 0191 1 GLOBAL ROUTINE PAS$$GET_UNSIGNED (      Get unsigned number string
131 0192 1      PFV: REF $PAS$PFV FILE_VARIABLE,    Pascal File Variable
132 0193 1      IN_FCB: REF $PAS$FCB_CONTROL_BLOCK;  File control block
133 0194 1      STRING_ADDR,                        Output string address
134 0195 1      STRING_LEN,                        Output string length
135 0196 1      FCB: REF $PAS$FCB_CONTROL_BLOCK     File control block
136 0197 1      ) : JSB_READ_UTIL =
137 0198 1
138 0199 1 ++
139 0200 1 FUNCTIONAL DESCRIPTION:
140 0201 1
141 0202 1     This procedure advances the textfile referenced by FCB until it
142 0203 1     locates a string that satisfies the Pascal UNSIGNED datatype
143 0204 1     syntax. The address and length of that string are returned as
144 0205 1     output parameters.
145 0206 1
146 0207 1 CALLING SEQUENCE:
147 0208 1
148 0209 1     Valid.wc.v = JSB_READ_UTIL PAS$$GET_UNSIGNED (PFV.mr.r, IN_FCB.mr.r;
149 0210 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
150 0211 1
151 0212 1 FORMAL PARAMETERS:
152 0213 1
153 0214 1     PFV          - The Pascal File Variable of the file.
154 0215 1
155 0216 1     IN_FCB       - The File Control Block of the file being scanned.
156 0217 1     It is assumed to be a textfile.
157 0218 1
158 0219 1     STRING_ADDR  - Output register parameter which is set to the
159 0220 1     address of the first byte of the string.
160 0221 1
161 0222 1     STRING_LEN   - Output register parameter which is set to the
162 0223 1     length of the string in bytes.
163 0224 1
164 0225 1     FCB          - Output register parameter which is the same as IN_FCB.
165 0226 1
166 0227 1 IMPLICIT INPUTS:
167 0228 1
168 0229 1     It is assumed that lazy-lookahead is not in progress.
169 0230 1
170 0231 1 IMPLICIT OUTPUTS:
171 0232 1
172 0233 1     FCB$A_RECORD_CUR points to the next character after the string, or
173 0234 1     EOL.
174 0235 1
175 0236 1 ROUTINE VALUE:
176 0237 1
177 0238 1     1 if string is a valid unsigned, 0 otherwise
178 0239 1     If 0 is returned, the pointer and length include the first bad character.
179 0240 1
180 0241 1 SIDE EFFECTS:
181 0242 1
182 0243 1     NONE
183 0244 1
184 0245 1 SIGNALLED ERRORS:
185 0246 1
```



```
186 0247 1 | NONE
187 0248 1 |
188 0249 1 | --
189 0250 1 |
190 0251 2 BEGIN
191 0252 2
192 0253 2 LOCAL
193 0254 2 CHAR; ! Character read
194 0255 2
195 0256 2
196 0257 2 | + Declare CHAR_BYTE which is the same as CHAR except that we can
197 0258 2 | test it as a signed byte. We want to leave CHAR as a longword
198 0259 2 | so that it can be used efficiently as an index.
199 0260 2 | -
200 0261 2
201 0262 2 BIND
202 0263 2 CHAR_BYTE = CHAR: BYTE SIGNED;
203 0264 2
204 0265 2 | +
205 0266 2 | Find first character that is not a blank or a tab, possibly skipping
206 0267 2 | records.
207 0268 2 | -
208 0269 2
209 0270 2 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
210 0271 2
211 0272 2 | +
212 0273 2 | At this point, CHAR contains the first character which is not a blank
213 0274 2 | or a tab. Initialize STRING_ADDR.
214 0275 2 | -
215 0276 2
216 0277 2 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
217 0278 2
218 0279 2 | +
219 0280 2 | In a loop, classify the characters until end-of-line or an invalid
220 0281 2 | character is found.
221 0282 2 | -
222 0283 2
223 0284 2 WHILE 1 DO
224 0285 2 BEGIN
225 0286 2
226 0287 2 | +
227 0288 2 | Screen out characters 128-255, which are not in CLASSTAB, by
228 0289 2 | doing a signed byte test for a negative value.
229 0290 2 | -
230 0291 2
231 0292 2 IF .CHAR_BYTE LSS 0
232 0293 2 THEN
233 0294 2 EXITLOOP;
234 0295 2
235 0296 2 | +
236 0297 2 | If the character is not a digit, exit.
237 0298 2 | -
238 0299 2
239 0300 2 IF .CLASSTAB [.CHAR] NEQU CLASS_DG
240 0301 2 THEN
241 0302 2 EXITLOOP;
242 0303 2
```

```
243 0304      !+
244 0305      !- Get another character if not at end-of-line.
245 0306
246 0307
247 0308      FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
248 0309      IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
249 0310      THEN
250 0311          CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
251 0312      ELSE
252 0313          EXITLOOP;
253 0314
254 0315      END;      ! Of WHILE loop
255 0316
256 0317      !+
257 0318      !- Set STRING_LEN to length of string and return success or failure
258 0319      !- depending on whether or not string is a valid unsigned.
259 0320
260 0321
261 0322      STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
262 0323      IF .STRING_LEN NEQ 0
263 0324      THEN
264 0325          RETURN 1;
265 0326
266 0327      STRING_LEN = 1;      ! Include first erroneous character
267 0328      RETURN 0;          ! Return failure
268 0329
269 0330      END;      ! End of routine PAS$$GET_UNSIGNED
```

.TITLE PAS\$\$READ_UTIL Utility routines used by READ
.IDENT \1-001\

.PSECT _PASSCODE,NOWRT, SHR, PIC,2

```
00 00 00 00 00 01 00 00 00 00 00 00 00 00 0000 CLASSTAB:
00 00 00 00 00 00 00 00 06 00 00 00 00 00 0000F
00 04 00 00 00 00 00 00 06 00 00 00 01 00 00001E
00 00 02 02 02 02 02 02 02 02 02 02 00 03 04 00002D
06 06 06 06 06 05 05 06 06 06 00 00 00 00 00003C
06 06 06 06 06 06 06 06 05 06 06 06 06 06 06 00004B
06 06 06 05 05 06 06 06 00 06 00 00 00 00 06 00005A
06 06 06 06 06 06 05 06 06 06 06 06 06 06 06 000069
00 00 00 00 00 06 06 06 06 06 06 06 06 06 000078
```

```
.BYTE 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, -
0, 4, 0, 0, 4, 3, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, -
2, 2, 0, 0, 0, 0, 0, 0, 0, 6, 6, 6, 5, 5, 5, 5, -
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, -
6, 6, 6, 6, 6, 6, 0, 0, 0, 0, 0, 6, 0, 6, 6, 6, -
6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, -
6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 0, 0, 0, 0, -
0, 0
```

.EXTRN PAS\$\$GET_UNSIGNED
.EXTRN PAS\$\$GET_INTEGER
.EXTRN PAS\$\$GET_REAL, PAS\$\$GET_ENUMERATED

0000V 30 00000 PAS\$\$GET UNSIGNED::

```
54      EC      A7      D0      00003      BSBW      FIND NON_BLANK      : 0270
50      95      00007      1$:      MOVL      -20(FCB), STRING_ADDR      : 0277
18      19      00009      TSTB      CHAR_BYTE      : 0292
02      FF70      CF40      91      0000B      BLSS      2$      : 0300
10      12      00011      CMPB      CLASSTAB[CHAR], #2
RNEQ      2$
```


PAS\$\$READ_UTIL		Utility routines used by READ		F 5		16-Sep-1984 01:55:25		VAX-11 Bliss-32 V4.0-742		Page 7	
1-001		PAS\$\$GET_UNSIGNED - Find an unsigned number str		14-Sep-1984 12:51:47				[PASRTL.SRC]PASREADUT.B32;1		(3)	

			EC	A7	D6	00013		INCL	-20(FCB)	:	0308
	F0	A7	EC	A7	D1	00016		CMPL	-20(FCB), -16(FCB)	:	0309
				06	1E	0001B		BGEQU	2\$:	
		50	EC	B7	9A	0001D		MOVZBL	@-20(FCB), CHAR	:	0311
				E4	11	00021		BRB	1\$:	
	55	EC	A7	54	C3	00023	2\$:	SUBL3	STRING_ADDR, -20(FCB), STRING_LEN	:	0322
				04	13	00028		BEQL	3\$:	0323
		50		01	D0	0002A		MOVL	#1, R0	:	0325
					05	0002D		RSB		:	
		55		01	D0	0002E	3\$:	MOVL	#1, STRING_LEN	:	0327
				50	D4	00031		CLRL	R0	:	0328
					05	00033		RSB		:	0330

; Routine Size: 52 bytes, Routine Base: _PAS\$CODE + 0080

:	270	0331	1	
:	271	0332	1	!<BLF/PAGE>

```
273 0333 1 ZSBTTL 'PAS$GET_INTEGER - Find a signed number string'
274 0334 1 GLOBAL ROUTINE PAS$GET_INTEGER (      Get signed number string
275 0335 1      PFV: REF $PAS$PFV_FILE_VARIABLE,    Pascal File Variable
276 0336 1      IN_FCB: REF $PAS$FCB_CONTROL_BLOCK;   File control block
277 0337 1      STRING_ADDR,                        Output string address
278 0338 1      STRING_LEN,                         Output string length
279 0339 1      FCB: REF $PAS$FCB_CONTROL_BLOCK      File control block
280 0340 1      ) : JSB_READ_UTIL =
281 0341 1
282 0342 1 ++
283 0343 1 FUNCTIONAL DESCRIPTION:
284 0344 1
285 0345 1     This procedure advances the textfile referenced by FCB until it
286 0346 1     locates a string that satisfies the Pascal INTEGER datatype
287 0347 1     syntax. The address and length of that string are returned as
288 0348 1     output parameters.
289 0349 1
290 0350 1 CALLING SEQUENCE:
291 0351 1
292 0352 1     Valid.wc.v = JSB PAS$GET_INTEGER (PFV.mr.r, IN_FCB.mr.r;
293 0353 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
294 0354 1
295 0355 1 FORMAL PARAMETERS:
296 0356 1
297 0357 1     PFV          - Pascal File Variable of the file.
298 0358 1
299 0359 1     IN_FCB       - The File Control Block of the file being scanned.
300 0360 1     It is assumed to be a textfile.
301 0361 1
302 0362 1     STRING_ADDR  - Output register parameter which is set to the
303 0363 1     address of the first byte of the string.
304 0364 1
305 0365 1     STRING_LEN   - Output register parameter which is set to the
306 0366 1     length of the string in bytes.
307 0367 1
308 0368 1     FCB          - Output register parameter which is the same as IN_FCB.
309 0369 1
310 0370 1 IMPLICIT INPUTS:
311 0371 1
312 0372 1     It is assumed that lazy-lookahead is not in progress.
313 0373 1
314 0374 1 IMPLICIT OUTPUTS:
315 0375 1
316 0376 1     FCB$A_RECORD_CUR points to the next character after the string, or
317 0377 1     EOL.
318 0378 1
319 0379 1 ROUTINE VALUE:
320 0380 1
321 0381 1     NONE
322 0382 1
323 0383 1 SIDE EFFECTS:
324 0384 1
325 0385 1     1 if string is a valid integer, 0 otherwise.
326 0386 1     If failure is returned, STRING_LEN includes the first bad character.
327 0387 1
328 0388 1 SIGNALLED ERRORS:
329 0389 1
```

```
330 0390 1 1 NONE
331 0391 1 1
332 0392 1 1
333 0393 1 1
334 0394 1 1 BEGIN
335 0395 1 1
336 0396 1 1 LOCAL
337 0397 1 1 CHAR, ! Character read
338 0398 1 1 VALID; ! 1 if string a valid unsigned.
339 0399 1 1
340 0400 1 1
341 0401 1 1 !+
342 0402 1 1 ! Declare CHAR_BYTE which is the same as CHAR except that we can
343 0403 1 1 ! test it as a signed byte. We want to leave CHAR as a longword
344 0404 1 1 ! so that it can be used efficiently as an index.
345 0405 1 1
346 0406 1 1 BIND
347 0407 1 1 CHAR_BYTE = CHAR: BYTE SIGNED;
348 0408 1 1
349 0409 1 1 !+
350 0410 1 1 ! Find first character that is not a blank or a tab, possibly skipping
351 0411 1 1 ! records.
352 0412 1 1
353 0413 1 1
354 0414 1 1 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
355 0415 1 1
356 0416 1 1 !+
357 0417 1 1 ! Initially, string is invalid.
358 0418 1 1
359 0419 1 1
360 0420 1 1 VALID = 0;
361 0421 1 1
362 0422 1 1 !+
363 0423 1 1 ! At this point, CHAR contains the first character which is not a blank
364 0424 1 1 ! or a tab. Initialize STRING_ADDR.
365 0425 1 1
366 0426 1 1
367 0427 1 1 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
368 0428 1 1
369 0429 1 1 !+
370 0430 1 1 ! If first character is a sign, advance pointer.
371 0431 1 1
372 0432 1 1
373 0433 1 1 IF .CHAR_BYTE GEQ 0
374 0434 1 1 THEN
375 0435 1 1 IF .CLASSTAB [.CHAR] EQLU CLASS_SI
376 0436 1 1 THEN
377 0437 1 1 BEGIN
378 0438 1 1 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
379 0439 1 1 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
380 0440 1 1 THEN
381 0441 1 1 CHAR = CHRCHAR (.FCB [FCB$A_RECORD_CUR])
382 0442 1 1 ELSE
383 0443 1 1 CHAR = %C' '; ! End of line
384 0444 1 1
385 0445 1 1 END;
386 0446 1 1 !+
```



```
387 0447 2 ! In a loop, classify the characters until end-of-line or an invalid
388 0448 ! character is found.
389 0449 !
390 0450 !
391 0451 WHILE 1 DO
392 0452 BEGIN
393 0453
394 0454 !+
395 0455 ! If the character's value is greater than or equal to 128,
396 0456 ! it can't possibly be valid, so exit. Do this by a test for
397 0457 ! negative on CHAR_BYTE.
398 0458 !-
399 0459
400 0460 IF .CHAR_BYTE LSS 0
401 0461 THEN
402 0462 EXITLOOP;
403 0463
404 0464 !+
405 0465 ! If the character is not a digit, exit.
406 0466 !-
407 0467
408 0468 IF .CLASSTAB [.CHAR] NEQU CLASS_DG
409 0469 THEN
410 0470 EXITLOOP;
411 0471
412 0472 !+
413 0473 ! At least one digit seen, so indicate string valid.
414 0474 !-
415 0475
416 0476 VALID = 1;
417 0477
418 0478 !+
419 0479 ! Get another character if not at end-of-line.
420 0480 !-
421 0481
422 0482 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
423 0483 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
424 0484 THEN
425 0485 CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
426 0486 ELSE
427 0487 EXITLOOP;
428 0488
429 0489 END; ! Of WHILE loop
430 0490
431 0491 !+
432 0492 ! Set STRING_LEN to length of string and return.
433 0493 !-
434 0494
435 0495 STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
436 0496 IF .STRING_LEN EQL 0 ! If so, VALID must be zero
437 0497 THEN
438 0498 STRING_LEN = 1;
439 0499 RETURN (.VALID);
440 0500
441 0501 END; ! End of routine PAS$$GET_INTEGER
```

```
0000V 30 00000 PAS$$GET_INTEGER::
      52 D4 00003 BSBW FIND_NON_BLANK      : 0414
51      A7 9E 00005 CLRL VALID                : 0420
      EC 61 D0 00009 MOVAB -20(FCB), R1        : 0427
54      50 95 0000C MOVL (R1), STRING_ADDR    :
      13 19 0000E TSTB CHAR_BYTE              : 0433
      04 FF37 CF40 91 00010 BLSS 1$            :
      08 12 00016 CMPB CLASSTAB[CHAR], #4     : 0435
      61 D6 00018 BNEQ 1$                      :
FO      A7 61 D1 0001A INCL (R1)                : 0438
      1A 1F 0001E CMPL (R1), -16(FCB)         : 0439
50      20 D0 00020 BLSSU 2$                   :
      50 95 00023 1$: TSTB CHAR_BYTE           : 0443
      19 19 00025 BLSS 3$                      : 0460
      02 FF20 CF40 91 00027 CMPB CLASSTAB[CHAR], #2 : 0468
      11 12 0002D BNEQ 3$                      :
52      01 D0 0002F MOVL #1, VALID              : 0476
      61 D6 00032 INCL (R1)                    : 0482
FO      A7 61 D1 00034 CMPL (R1), -16(FCB)     : 0483
      06 1E 00038 BGEQU 3$                     :
50      00 B1 9A 0003A 2$: MOVZBL 20(R1), CHAR : 0485
      E3 11 0003E BRB 1$                       :
      55 54 C3 00040 3$: SUBL3 STRING_ADDR, (R1), STRING_LEN : 0495
      03 12 00044 BNEQ 4$                      : 0496
55      01 D0 00046 MOVL #1, STRING_LEN        : 0498
50      52 D0 00049 4$: MOVL VALID, R0         : 0499
      05 0004C RSB                             : 0501
```

: Routine Size: 77 bytes, Routine Base: _PAS\$CODE + 00B4

: 442 0502 1
: 443 0503 1 !<BLF/PAGE>

```
445 0504 1 XSBTTL 'PAS$GET_REAL - Find a real number string'
446 0505 1 GLOBAL ROUTINE PAS$GET_REAL (      Get real number string
447 0506 1     PFV: REF $PASSPFV_FILE_VARIABLE,  Pascal File Variable
448 0507 1     IN_FCB: REF $PASSFCB_CONTROL_BLOCK; File control block
449 0508 1     STRING_ADDR,      Output string address
450 0509 1     STRING_LEN,      Output string length
451 0510 1     FCB: REF $PASSFCB_CONTROL_BLOCK  File control block
452 0511 1 ) : JSB_READ_UTIL =
453 0512 1
454 0513 1 **
455 0514 1 FUNCTIONAL DESCRIPTION:
456 0515 1
457 0516 1     This procedure advances the textfile referenced by FCB until it
458 0517 1     locates a string that satisfies the Pascal REAL datatype
459 0518 1     syntax. The address and length of that string are returned as
460 0519 1     output parameters.
461 0520 1
462 0521 1 CALLING SEQUENCE:
463 0522 1
464 0523 1     Valid.wc.v = JSB PAS$GET_REAL (PFV.mr.r, IN_FCB.mr.r;
465 0524 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
466 0525 1
467 0526 1 FORMAL PARAMETERS:
468 0527 1
469 0528 1     PFV          - Pascal File Variable for the file.
470 0529 1
471 0530 1     IN_FCB       - The File Control Block of the file being scanned.
472 0531 1     It is assumed to be a textfile.
473 0532 1
474 0533 1     STRING_ADDR  - Output register parameter which is set to the
475 0534 1     address of the first byte of the string.
476 0535 1
477 0536 1     STRING_LEN   - Output register parameter which is set to the
478 0537 1     length of the string in bytes.
479 0538 1
480 0539 1     FCB          - Output register parameter which is the same as IN_FCB.
481 0540 1
482 0541 1 IMPLICIT INPUTS:
483 0542 1
484 0543 1     It is assumed that lazy-lookahead is not in progress.
485 0544 1
486 0545 1 IMPLICIT OUTPUTS:
487 0546 1
488 0547 1     FCB$A_RECORD_CUR points to the next character after the string, or
489 0548 1     EOL.
490 0549 1
491 0550 1 ROUTINE VALUE:
492 0551 1
493 0552 1     1 if string is a valid real, 0 otherwise.
494 0553 1     If failure is returned, STRING_LEN includes the first bad character
495 0554 1
496 0555 1 SIDE EFFECTS:
497 0556 1
498 0557 1     NONE
499 0558 1
500 0559 1 SIGNALLED ERRORS:
501 0560 1
```



```
0561 1 | NONE
0562 1 |
0563 1 |
0564 1 |
0565 2 BEGIN
0566 2
0567 2 LOCAL
0568 2 CHAR, | Character read
0569 2 FLAGS: BITVECTOR [5]; | Indicate value fields seen
0570 2
0571 2 |
0572 2 | + Declare CHAR_BYTE which is the same as CHAR except that we can
0573 2 | test it as a signed byte. We want to leave CHAR as a longword
0574 2 | so that it can be used efficiently as an index.
0575 2 |
0576 2 |
0577 2 BIND
0578 2 CHAR_BYTE = CHAR: BYTE SIGNED;
0579 2
0580 2 LITERAL
0581 2 FLAGS_EXPLT = 0, | Exponent letter seen
0582 2 FLAGS_POINT = 1, | Decimal point seen
0583 2 FLAGS_FRADG = 2, | Fraction digit seen
0584 2 FLAGS_EXPDG = 3, | Exponent digit seen
0585 2 FLAGS_EXPSI = 4; | Exponent sign seen
0586 2
0587 2 |
0588 2 | + Find first character that is not a blank or a tab, possibly skipping
0589 2 | records.
0590 2 |
0591 2 |
0592 2 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
0593 2
0594 2 |
0595 2 | + Initialize local flags.
0596 2 |
0597 2 |
0598 2 FLAGS = 0;
0599 2
0600 2 |
0601 2 | + At this point, CHAR contains the first character which is not a blank
0602 2 | or a tab. Initialize STRING_ADDR.
0603 2 |
0604 2 |
0605 2 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
0606 2
0607 2 |
0608 2 | + If first character is a sign, advance pointer.
0609 2 |
0610 2 |
0611 2 IF .CHAR_BYTE GEQ 0
0612 2 THEN
0613 2 IF .CLASSTAB [.CHAR] EQLU CLASS_SI
0614 2 THEN
0615 2 BEGIN
0616 2 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
0617 2 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
```

```
559 0618 3
560 0619
561 0620 THEN CHAR = CHRCHAR (.FCB [FCB$A_RECORD_CUR])
562 0621 ELSE CHAR = %C' '; ! End of Line
563 0622 END;
564 0623
565 0624
566 0625 !+ In a loop, classify the characters until end-of-line or an invalid
567 0626 character is found.
568 0627
569 0628
570 0629 WHILE 1 DO
571 0630 BEGIN
572 0631
573 0632 !+
574 0633 If the character's value is greater than or equal to 128,
575 0634 it can't possibly be valid, so exit. Do this with a signed test
576 0635 on CHAR_BYTE.
577 0636
578 0637
579 0638 IF .CHAR_BYTE LSS 0
580 0639 THEN
581 0640 EXITLOOP;
582 0641
583 0642 !+
584 0643 Select action based on character class.
585 0644
586 0645
587 0646 CASE .CLASSTAB [.CHAR] FROM CLASS_IV TO CLASS_LT OF
588 0647
589 0648 SET
590 0649
591 0650 [CLASS_DG]: ! Digit, always valid
592 0651 BEGIN
593 0652 IF .FLAGS [FLAGS_EXPLT] ! Exponent letter already seen?
594 0653 THEN
595 0654 BEGIN
596 0655 FLAGS [FLAGS_EXPSI] = 1; ! Prohibit future signs
597 0656 FLAGS [FLAGS_EXPDG] = 1; ! Mark exponent digit seen
598 0657 END
599 0658 ELSE
600 0659 FLAGS [FLAGS_FRADG] = 1; ! Mark fraction digit seen
601 0660 END;
602 0661
603 0662 [CLASS_SI]: ! Sign character
604 0663 BEGIN
605 0664 IF NOT .FLAGS [FLAGS_EXPLT] ! Exponent letter not seen?
606 0665 THEN
607 0666 EXITLOOP; ! If so, invalid
608 0667 IF .FLAGS [FLAGS_EXPSI] ! Exponent sign seen?
609 0668 THEN
610 0669 EXITLOOP; ! If so, invalid
611 0670 FLAGS [FLAGS_EXPSI] = 1; ! Indicate exponent sign seen
612 0671 END;
613 0672
614 0673 [CLASS_EL]: ! Exponent letter
615 0674 BEGIN
```

```
616      0675 4      IF .FLAGS [FLAGS_EXPLT]      ! Exponent letter already seen?
617      0676 4      THEN
618      0677 4          EXITLOOP;                ! If so, invalid
619      0678 4      IF NOT .FLAGS [FLAGS_FRADG]    ! Fraction digit seen?
620      0679 4      THEN
621      0680 4          EXITLOOP;                ! If not, invalid
622      0681 4      FLAGS [FLAGS_EXPLT] = 1;        ! Mark exponent letter seen
623      0682 4      FLAGS [FLAGS_POINT] = 1;        ! Prohibit future decimal point
624      0683 4      END;
625      0684 4
626      0685 3      [CLASS DP]:                ! Decimal point
627      0686 4      BEGIN
628      0687 4      IF .FLAGS [FLAGS_POINT]        ! Decimal point already seen?
629      0688 4      THEN
630      0689 4          EXITLOOP;                ! If so, invalid
631      0690 4      FLAGS [FLAGS_POINT] = 1;        ! Mark decimal point seen
632      0691 4      END;
633      0692 4
634      0693 4      [INRANGE, OTRANGE]:
635      0694 4          EXITLOOP;                ! Invalid
636      0695 4
637      0696 4      TES;
638      0697 4
639      0698 4
640      0699 4      !+ Get another character if not at end-of-line.
641      0700 4      !-
642      0701 4
643      0702 4      FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
644      0703 4      IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
645      0704 4      THEN
646      0705 4          CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
647      0706 4      ELSE
648      0707 4          EXITLOOP;
649      0708 4
650      0709 4      END;                ! Of WHILE loop
651      0710 4
652      0711 4
653      0712 4      !+ Set STRING_LEN to length of string and return function value indicating
654      0713 4      !- whether or not string is valid.
655      0714 4
656      0715 4
657      0716 4      STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
658      0717 4      IF .STRING_LEN EQL 0                ! If so, string is invalid
659      0718 4      THEN
660      0719 4          STRING_LEN = 1;
661      0720 4      RETURN (
662      0721 4          IF .FLAGS [FLAGS_FRADG] AND      ! Fraction digit required
663      0722 4          ((NOT .FLAGS [FLAGS_EXPLT]) OR .FLAGS [FLAGS_EXPDG]) ! If exponent, must have digits
664      0723 4      THEN
665      0724 4          1      ! Valid
666      0725 4      ELSE
667      0726 4          0      ! Invalid
668      0727 4      );
669      0728 4
670      0729 4      END;                ! End of routine PASS$GET_REAL
```


			0000V	30	00000	PAS\$\$GET	REAL::		
				52	94	00003	BSBW	FIND_NON_BLANK	0592
							CLRB	FLAGS	0598
		51	EC	A7	9E	00005	MOVAB	-20(FCB), R1	0605
		54		61	D0	00009	MOVL	(R1), STRING_ADDR	
				50	95	0000C	TSTB	CHAR_BYTE	0611
				13	19	0000E	BLSS	1\$	
		04	FEEA	CF40	91	00010	CMPB	CLASSTAB[CHAR], #4	0613
				0B	12	00016	BNEQ	1\$	
				61	D6	00018	INCL	(R1)	0616
		FO	A7	61	D1	0001A	CMPL	(R1), -16(FCB)	0617
				52	1F	0001E	BLSSU	10\$	
		50		20	D0	00020	MOVL	#32, CHAR	0621
				50	95	00023	TSTB	CHAR_BYTE	0638
				51	19	00025	BLSS	11\$	
			FED3	CF40	8F	00027	CASEB	CLASSTAB[CHAR], #0, #6	0646
0035	06	00		004A		0002E	.WORD	11\$-2\$,-	
	0010	004A		0029	001D	00036		11\$-2\$,-	
								3\$-2\$,-	
								7\$-2\$,-	
								5\$-2\$,-	
								6\$-2\$,-	
								11\$-2\$	
				3A	11	0003C	BRB	11\$	0694
		05		52	E9	0003E	BLBC	FLAGS, 4\$	0652
		52		18	88	00041	BISB2	#24, FLAGS	0656
				24	11	00044	BRB	9\$	0652
		52		04	88	00046	BISB2	#4, FLAGS	0659
				1F	11	00049	BRB	9\$	0646
		2A		52	E9	0004B	BLBC	FLAGS, 11\$	0664
26		52		04	E0	0004E	BBS	#4, FLAGS, 11\$	0667
		52		10	88	00052	BISB2	#16, FLAGS	0670
				13	11	00055	BRB	9\$	0646
		1E		52	E8	00057	BLBS	FLAGS, 11\$	0675
1A		52		02	E1	0005A	BBC	#2, FLAGS, 11\$	0678
		52		01	88	0005E	BISB2	#1, FLAGS	0681
				04	11	00061	BRB	8\$	0682
11		52		01	E0	00063	BBS	#1, FLAGS, 11\$	0687
		52		02	88	00067	BISB2	#2, FLAGS	0690
				61	D6	0006A	INCL	(R1)	0702
		FO	A7	61	D1	0006C	CMPL	(R1), -16(FCB)	0703
				06	1E	00070	BGEQU	11\$	
		50		B1	9A	00072	MOVZBL	20(R1), CHAR	0705
			00	AB	11	00076	BRB	1\$	
55		61		54	C3	00078	SUBL3	STRING_ADDR, (R1), STRING_LEN	0716
				03	12	0007C	BNEQ	12\$	0717
		55		01	D0	0007E	MOVL	#1, STRING_LEN	0719
0B		52		02	E1	00081	BBC	#2, FLAGS, -14\$	0721
		04		52	E9	00085	BLBC	FLAGS, 13\$	0722
04		52		03	E1	00088	BBC	#3, FLAGS, 14\$	
				01	D0	0008C	MOVL	#1, R0	0721
				05	0008F	RSB			
				50	D4	00090	CLRL	R0	0729
				05	00092	RSB			

PAS\$\$READ_UTIL Utility routines used by READ
1-001 PAS\$\$GET_REAL - Find a real number string

C 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 17
(5)

: Routine Size: 147 bytes, Routine Base: _PAS\$CODE + 0101

: 671 0730 1
: 672 0731 1 !<BLF/PAGE>

```
674 0732 1 %SBTTL 'PASS$GET_ENUMERATED - Find an enumerated value string'
675 0733 1 GLOBAL ROUTINE PASS$GET_ENUMERATED (      Get enumerated value string
676 0734 1      PFV: REF $PASS$PFV_FILE_VARIABLE,      Pascal File Variable
677 0735 1      IN_FCB: REF $PASS$FCB_CONTROL_BLOCK;      File control block
678 0736 1      STRING_ADDR,      Output string address
679 0737 1      STRING_LEN,      Output string length
680 0738 1      FCB: REF $PASS$FCB_CONTROL_BLOCK      File control block
681 0739 1      ) : JSB_READ_UTIL =
682 0740 1
683 0741 1 ++
684 0742 1 FUNCTIONAL DESCRIPTION:
685 0743 1
686 0744 1     This procedure advances the textfile referenced by FCB until it
687 0745 1     locates a string that satisfies the Pascal enumerated type value
688 0746 1     syntax. The address and length of that string are returned as
689 0747 1     output parameters.
690 0748 1
691 0749 1 CALLING SEQUENCE:
692 0750 1
693 0751 1     Valid.wc.v = JSB PASS$GET_ENUMERATED (PFV.mr.r, IN_FCB.mr.r;
694 0752 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
695 0753 1
696 0754 1 FORMAL PARAMETERS:
697 0755 1
698 0756 1     PFV      - Pascal File Variable for the file.
699 0757 1
700 0758 1     IN_FCB   - The File Control Block of the file being scanned.
701 0759 1     It is assumed to be a textfile.
702 0760 1
703 0761 1     STRING_ADDR - Output register parameter which is set to the
704 0762 1     address of the first byte of the string.
705 0763 1
706 0764 1     STRING_LEN - Output register parameter which is set to the
707 0765 1     length of the string in bytes.
708 0766 1
709 0767 1     FCB      - Output register parameter which is the same as IN_FCB.
710 0768 1
711 0769 1 IMPLICIT INPUTS:
712 0770 1
713 0771 1     It is assumed that lazy-lookahead is not in progress.
714 0772 1
715 0773 1 IMPLICIT OUTPUTS:
716 0774 1
717 0775 1     FCB$A_RECORD_CUR points to the next character after the string, or
718 0776 1     EOL.
719 0777 1
720 0778 1 ROUTINE VALUE:
721 0779 1
722 0780 1     1 if string is a valid enumerated value, 0 otherwise
723 0781 1     If failure is returned, STRING_LEN includes the first bad character
724 0782 1
725 0783 1 SIDE EFFECTS:
726 0784 1
727 0785 1     NONE
728 0786 1
729 0787 1 SIGNALLED ERRORS:
730 0788 1
```



```
731 0789 1 | NONE
732 0790 1 |
733 0791 1 | --
734 0792 1 |
735 0793 2 | BEGIN
736 0794 2 |
737 0795 2 | LOCAL
738 0796 2 | CHAR,
739 0797 2 | VALID_CHAR_MASK: BITVECTOR [32];
740 0798 2 |
741 0799 2 |
742 0C00 2 |
743 0801 2 |
744 0802 2 |
745 0803 2 |
746 0804 2 |
747 0805 2 |
748 0806 2 |
749 0807 2 |
750 0808 2 |
751 0809 2 |
752 0810 2 |
753 0811 2 |
754 0812 2 |
755 0813 2 |
756 0814 2 |
757 0815 2 |
758 0816 2 |
759 0817 2 |
760 0818 2 |
761 0819 2 |
762 0820 2 |
763 0821 2 |
764 0822 2 |
765 0823 2 |
766 0824 2 |
767 0825 2 |
768 0826 2 |
769 0827 2 |
770 0828 2 |
771 0829 2 |
772 0830 2 |
773 0831 2 |
774 0832 2 |
775 0833 2 |
776 0834 2 |
777 0835 2 |
778 0836 2 |
779 0837 2 |
780 0838 2 |
781 0839 2 |
782 0840 2 |
783 0841 2 |
784 0842 2 |
785 0843 2 |
786 0844 2 |
787 0845 2 |
```

NONE

BEGIN

LOCAL

CHAR,

VALID_CHAR_MASK: BITVECTOR [32];

Character read
Bit is set if associated
character class is valid
at this point.

+
Declare CHAR_BYTE which is the same as CHAR except that we can
test it as a signed byte. We want to leave CHAR as a longword
so that it can be used efficiently as an index.
-

BIND

CHAR_BYTE = CHAR: BYTE SIGNED;

+
Find first character that is not a blank or a tab, possibly skipping
records.
-

CHAR = FIND_NON_BLANK (PFV [PFVSR_PFV], IN_FCB [FCBSR_FCB]; FCB);

+
At this point, CHAR contains the first character which is not a blank
or a tab. Initialize STRING_ADDR.
-

STRING_ADDR = .FCB [FCBSA_RECORD_CUR];

+
First character must be a letter. (Class LT excludes exponent
letters, so add class EL).
-

VALID_CHAR_MASK = (1^CLASS_LT)+(1^CLASS_EL);

+
In a loop, classify the characters until end-of-line or an invalid
character is found.
-

WHILE 1 DO

BEGIN

+
If the character's value is greater than or equal to 128,
it can't possibly be valid, so exit. Do this with a signed test
on CHAR_BYTE.
-

```
788 0846 IF .CHAR_BYTE LSS 0
789 0847 THEN
790 0848     EXITLOOP;
791 0849
792 0850
793 0851     !+ Get the class of the character from CLASSTAB and test its
794 0852     ! corresponding bit in VALID_CHAR_MASK. If it is not set, that
795 0853     ! character is not acceptable.
796 0854     !-
797 0855
798 0856 IF NOT .VALID_CHAR_MASK [.CLASSTAB [.CHAR]]
799 0857 THEN
800 0858     EXITLOOP;
801 0859
802 0860
803 0861     !+ Allow digits to appear from now on.
804 0862     !-
805 0863
806 0864     VALID_CHAR_MASK [CLASS_DG] = 1;
807 0865
808 0866
809 0867     !+ Get another character if not at end-of-line.
810 0868     !-
811 0869
812 0870     FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
813 0871     IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
814 0872     THEN
815 0873         CHAR = CHRCHAR (.FCB [FCB$A_RECORD_CUR])
816 0874     ELSE
817 0875         EXITLOOP;
818 0876
819 0877     END;      ! Of WHILE loop
820 0878
821 0879
822 0880
823 0881     !+ Set STRING_LEN to length of string and return function value indicating
824 0882     ! whether or not string is valid.
825 0883     !-
826 0884
827 0885     STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
828 0886     IF .STRING_LEN NEQ 0
829 0887     THEN
830 0888         RETURN 1;
831 0889
832 0890     STRING_LEN = 1;      ! Include first had character
833 0891     RETURN 0;           ! Failure
834 0892
835 0893     END;
                                ! End of routine PAS$GET_ENUMERATED
```

0000V 30 0000 PAS\$GET_ENUMERATED::

54 EC A7 D0 00003
51 60 8F 9A 00007BSBW FIND_NON_BLANK
MOVL -20(FCB), STRING_ADDR
MOVZBL #96, VALID_CHAR_MASK: 0815
: 0822
: 0830

PASS\$READ_UTIL
1-001

Utility routines used by READ
PASS\$GET_ENUMERATED - Find an enumerated value

6 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 21
(6)

			50	95	0000B	1\$:	TSTB	CHAR_BYTE	:	0846
			1D	19	0000D		BLSS	2\$:	
13	52	FE5B	CF40	9A	0000F		MOVZBL	CLASSTAB[CHAR], R2	:	0856
	51		52	E1	00015		BBC	R2, VALID_CHAR_MASK, 2\$:	
	51		04	88	00019		BISB2	#4, VALID_CHAR_MASK	:	0864
		EC	A7	D6	0001C		INCL	-20(FCB)	:	0870
	FO	A7	EC	A7	D1	0001F	CMPL	-20(FCB), -16(FCB)	:	0871
			06	1E	00024		BGEQU	2\$:	
	50		EC	B7	9A	00026	MOVZBL	@-20(FCB), CHAR	:	0873
			DF	11	0002A		BRB	1\$:	
55	EC	A7	54	C3	0002C	2\$:	SUBL3	STRING_ADDR, -20(FCB), STRING_LEN	:	0885
			04	13	00031		BEOL	3\$:	0886
	50		01	D0	00033		MOVL	#1, R0	:	0888
				05	00036		RSB		:	
	55		01	D0	00037	3\$:	MOVL	#1, STRING_LEN	:	0890
			50	D4	0003A		CLRL	R0	:	0891
				05	0003C		RSB		:	0893

; Routine Size: 61 bytes, Routine Base: _PASS\$CODE + 0194

: 836 0894 1
: 837 0895 1 !<BLF/PAGE>

PAS
1-0


```
839 0896 1 $SBTTL 'FIND_NON_BLANK - Find first non-blank'
840 0897 1 ROUTINE FIND_NON_BLANK (
841 0898 1     PFV: REF $PASSPFV FILE VARIABLE,
842 0899 1     IN_FCB: REF $PASSFCB CONTROL_BLOCK;
843 0900 1     FCB: REF $PASSFCB CONTROL_BLOCK
844 0901 1 ) : JSB_FIND_NON_BLANK =
845 0902 1
846 0903 1 ++
847 0904 1 FUNCTIONAL DESCRIPTION:
848 0905 1
849 0906 1     This procedure advances the textfile referenced by FCB until it
850 0907 1     locates the first character which is not a blank or a tab. It
851 0908 1     returns that character as its function value.
852 0909 1
853 0910 1 CALLING SEQUENCE:
854 0911 1
855 0912 1     CHAR.wt.v = JSB_FIND_NON_BLANK (PFV.mr.r, IN_FCB.mr.r; FCB.mr.r)
856 0913 1
857 0914 1 FORMAL PARAMETERS:
858 0915 1
859 0916 1     PFV           - Pascal File Variable for the file.
860 0917 1
861 0918 1     IN_FCB        - The File Control Block of the file being scanned.
862 0919 1                   It is assumed to be a textfile.
863 0920 1
864 0921 1     FCB           - Output register parameter which is the same as IN_FCB.
865 0922 1
866 0923 1 IMPLICIT INPUTS:
867 0924 1
868 0925 1     It is assumed that lazy-lookahead is not in progress.
869 0926 1
870 0927 1 IMPLICIT OUTPUTS:
871 0928 1
872 0929 1     FCB$A_RECORD_CUR points to the found character.
873 0930 1
874 0931 1 ROUTINE VALUE:
875 0932 1
876 0933 1     The character found which is not a blank or a tab.
877 0934 1
878 0935 1 SIDE EFFECTS:
879 0936 1
880 0937 1     NONE
881 0938 1
882 0939 1 SIGNALLED ERRORS:
883 0940 1
884 0941 1     GETAFTEOF - GET after end-of-file
885 0942 1
886 0943 1 --
887 0944 1 BEGIN
888 0945 1
889 0946 1 LOCAL
890 0947 1 CHAR;
891 0948 1
892 0949 1
893 0950 1 ++
894 0951 1 ! Declare CHAR_BYTE which is the same as CHAR except that we can
895 0952 1 ! test it as a signed byte. We want to leave CHAR as a longword
```

```
896 0953 2 | so that it can be used efficiently as an index.
897 0954 2 |
898 0955 2 |
899 0956 2 BIND
900 0957 2 CHAR_BYTE = CHAR: BYTE SIGNED;
901 0958 2
902 0959 2 FCB = .IN_FCB;
903 0960 2
904 0961 2 |
905 0962 2 | Find first character that is not a blank or a tab, possibly skipping
906 0963 2 | records.
907 0964 2 |
908 0965 2 |
909 0966 2 WHILE 1 DO
910 0967 2 BEGIN
911 0968 2 |
912 0969 2 | If we are at end-of-line, get another record. This is done by
913 0970 2 | setting lazy-lookahead and then calling PAS$$LOOK_AHEAD.
914 0971 2 |
915 0972 2 |
916 0973 2 IF .FCB [FCB$A_RECORD_CUR] GEQA .FCB [FCB$A_RECORD_END]
917 0974 2 THEN
918 0975 2 BEGIN
919 0976 2 FCB [FCB$V_LAZY] = 1; ! Set lazy lookahead
920 0977 2 PAS$$LOOK_AHEAD (PFV [PFV$R_PFV], FCB [FCB$R_FCB]; FCB);
921 0978 2 END
922 0979 2 ELSE
923 0980 2 BEGIN
924 0981 2 |
925 0982 2 | Get next character, advancing pointer, and check class for blank
926 0983 2 | or tab.
927 0984 2 |
928 0985 2 CHAR = CH$RCHAR A (FCB [FCB$A_RECORD_CUR]);
929 0986 2 IF (.CHAR_BYTE [SS 0) OR (.CLASSTAB [.CHAR] NEQ CLASS_BT)
930 0987 2 THEN
931 0988 2 BEGIN
932 0989 2 |
933 0990 2 | Non blank/tab found. Reset record pointer to point
934 0991 2 | to character and exit loop.
935 0992 2 |
936 0993 2 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] - 1;
937 0994 2 EXITLOOP;
938 0995 2 END;
939 0996 2 END;
940 0997 2
941 0998 2 END; ! Of WHILE loop
942 0999 2
943 1000 2 RETURN .CHAR; ! Return found character
944 1001 2
945 1002 2 END; ! End of routine FIND_NON_BLANK
```

.EXTRN PAS\$\$LOOK_AHEAD

F0 A7 EC A7 D1 0000 FIND_NON_BLANK:
[MPL -20(FCB), -16(FCB)

: 0973

PASS\$READ_UTIL Utility routines used by READ
1-001 FIND_NON_BLANK - Find first non-blank

J 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 24
(7)

```

      FD  A7  00000000G  0C  1F 00005
                        04  88 00007
                        00  16 0000B
                        ED  11 00011
      58      EC  B7  9A 00013 1$:
                        EC  A7  D6 00017
                        58  95 0001A
                        08  19 0001C
      01      FEOC CF48 91 0001E
                        DA  13 00024
                        EC  A7  D7 00026 2$:
      50      58  D0 00029
                        05 0002C
```

```

BLSSU  1$
BISB2  #4, -3(FCB)
JSB    PASS$LOOK_AHEAD
BRB    FIND_NON_BLANK
MOVZBL @-20(FCB), CHAR
INCL   -20(FCB)
TSTB   CHAR_BYTE
BLSS   2$
CMPB   CLASSTAB[CHAR], #1
BEQL   FIND_NON_BLANK
DECL   -20(FCB)
MOVL   CHAR, R0
RSB
```

```

: 0976
: 0977
: 0973
: 0985
: 0986
:
: 0993
: 1000
: 1002
```

; Routine Size: 45 bytes, Routine Base: _PASS\$CODE + 01D1

```

: 946      1003 1
: 947      1004 1 !<BLF/PAGE>
```


PASS\$READ_UTIL Utility routines used by READ
1-001 FIND_NON_BLANK - Find first non-blank

K 6
16-Sep-1984 01:55:25 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:51:47 [PASRTL.SRC]PASREADUT.B32;1

Page 25
(8)

: 949 1005 1 END
: 950 1006 1
: 951 1007 0 ELUDOM

! End of module PASS\$READ_UTIL

PSECT SUMMARY

Name	Bytes	Attributes
_PASS\$CODE	510	NOVEC,NOWRT, RD , EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	0	0	581	00:01.0
\$255\$DUA28:[PASRTL.OBJ]PASLIB.L32;1	427	86	20	33	00:00.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LISS\$:PASREADUT/OBJ=OBJ\$:PASREADUT MSRC\$:PASREADUT/UPDATE=(ENH\$:PASREADUT)

: Size: 382 code + 128 data bytes
: Run Time: 00:15.4
: Elapsed Time: 00:53.6
: Lines/CPU Min: 3923
: Lexemes/CPU-Min: 15841
: Memory Used: 110 pages
: Compilation Complete

0296

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY